UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

André Seidel Oliveira

MAPEAMENTO EM GRADE DE OCUPAÇÃO UTILIZANDO REDES NEURAIS PROFUNDAS

Vitória-ES

Dezembro/2018

André Seidel Oliveira

MAPEAMENTO EM GRADE DE OCUPAÇÃO UTILIZANDO REDES NEURAIS PROFUNDAS

Parte manuscrita do Projeto de Graduação do aluno André Seidel Oliveira, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Alberto Ferreira De Souza.
Coorientador: MSc. Vinícius Brito Cardoso.

Vitória-ES

Dezembro/2018

André Seidel Oliveira

MAPEAMENTO EM GRADE DE OCUPAÇÃO UTILIZANDO REDES NEURAIS PROFUNDAS

Parte manuscrita do Projeto de Graduação do aluno André Seidel Oliveira, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Alberto Ferreira De Souza.
Coorientador: MSc. Vinícius Brito Cardoso.

Prof. Dr. Alberto Ferreira De Souza
Universidade Federal do Espírito Santo
Orientador

MSc. Vinícius Brito Cardoso
Universidade Federal do Espírito Santo
Coorientador

Prof. Dr. Jorge Leonid Aching Samatelo
Universidade Federal do Espírito Santo
Avaliador

MSc. Rânik Guidolini
Universidade Federal do Espírito Santo
Avaliador

André Seidel Oliveira
Universidade Federal do Espírito Santo
Aluno

Vitória-ES

Dezembro/2018

RESUMO

O aprendizado com redes neurais é um método associado à inteligência artificial que busca reproduzir o funcionamento do cérebro para gerar conhecimentos à partir de exemplos fornecidos. Esse tipo de algoritmo é generalizável para diversas aplicações, bastando que se tenha acesso a uma quantidade razoável de dados relacionados ao problema: Uma rede neural será tão boa quanto a quantidade e qualidade dos seus dados lhe permitir.
Seguindo a tendência atual de se buscar soluções com redes neurais para problemas de engenharia, este documento apresenta uma proposta de mapeamento de ocupação do entorno do carro autônomo IARA, da UFES. Utilizando o sensor laser LiDAR, espera-se reproduzir o mapa de ocupação gerado por métodos clássicos de robótica móvel, dessa vez por meio de redes neurais convolucionais.
Portanto, este trabalho tem por objetivo final comprovar a viabilidade da utilização de inteligência artificial, bem como mostrar diferenças de desempenho entre esse novo método de mapeamento e o atual utilizado no veículo.

Palavras-chave: Inteligência Artificial; Redes Neurais; Redes Neurais Convolucionais; Veículo autônomo; Mapeamento de obstáculos.

ABSTRACT

Neural networks is a machine learning method associated with the Artificial Intelligence field, that aim to reproduce brains mechanisms to generate knowledge from samples. This kind of algorithm can be generalized for plenty of applications, as long as one have access to a sufficient amount of data related to the subject: A neural network will be as good as the quantity and quality of the data set allows it to be.
Following the current trend of seeking solutions with neural networks for engineering, this document presents a new proposal of occupancy mapping for UFES autonomous car, IARA. Using the laser sensor LiDAR, it is expected to reproduce the occupancy map generated by classical mobile robotics algorithms, this time by the means of convolutional neural networks.
Therefore, this work has the final goal of proving the viability of AI to solve this sort of problem, and also to show performance differences between this new method and the current one.

Keywords: Artificial Intelligence; Neural Networks; Convolutional Neural Networks; Autonomous Driving; Obstacle Mapping.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

UFES	*Universidade Federal do Espírito Santo*
IARA	*Intelligent Autonomous Robotic Automobile*
LCAD	*Laboratório de Computação de Alto Desempenho*
LIDAR	*Light Detection and Ranging*
RAM	*Random access memory*
SLAM	*Simultaneous Localization and Mapping*
GPU	*Graphics Processing Unit*
OGM	*Occupancy Grid Mapping*
ms	*milissegundo*

LISTA DE SÍMBOLOS

p	*Probabilidade de uma variável aleatória*
w	*Peso sináptico da rede neural*
b	*Bias aditivo aplicado aos neurônios da rede neural*
t	*Tempo discretizado*
x	*Posição do robô*
m	*Mapa discreto de ocupação*
u	*Vetor de controle do robô*

SUMÁRIO

1 INTRODUÇÃO

A robótica móvel autônoma é uma área que têm crescido muito nos últimos vinte anos. Em 2004 foi promovida pela Agência de Projetos de Pesquisa Avançada de Defesa (DARPA) americana a primeira competição de veículos autônomos, com o objetivo de se realizar uma travessia pelo deserto sem intervenção humana (DARPA, 2005). Após essa, outras duas competições foram promovidas pela DARPA em 2005 e 2007 onde respectivamente cinco e seis veículos concluíram o circuito. Esses eventos ficaram conhecidos como os aceleradores do desenvolvimento dos carros autônomos. Enquanto na primeira edição desta competição nenhuma equipe conseguiu terminar o circuito, hoje os carros autônomos já são realidade e serviços com protótipos de direção autônoma são prestados ao grande público, por empresas como Google, Uber e Tesla.

Apesar dos grandes avanços já consolidados nesse tema, os veículos comerciais da atualidade ainda não tratam com exatidão todas as situações encontradas no trânsito nas cidades, como a presença de pedestres, sinalizações e outros carros. Por esse motivo, universidades e centros de pesquisa no Brasil e no mundo desenvolvem pesquisa para cada vez mais melhorar e aumentar a autonomia dessa tecnologia. Alguns dos grandes centros e empresas internacionais que realizam pesquisa nesse assunto são o MIT, Universidade de Stanford, Google e Uber. Já no Brasil entre as pesquisas na área podemos destacar a Universidade Federal do Espírito Santo (UFES) que através do Laboratório de Computação de Alto Desempenho (LCAD) tem tido grandes avanços com o projeto IARA, e a Universidade de São Paulo com o projeto CaRINA.

O robô móvel autônomo IARA (*Intelligent Autonomous Robotic Automobile*), mostrado na Figura 1, é desenvolvido desde 2009 pelo Laboratório de Computação de Alto Desempenho, da UFES. Esse veículo, que já realizou um percurso de 74 quilômetros autonomamente, utiliza diversos sensores para resolver os inúmeros desafios relacionados à direção autônoma. Os principais sensores utilizados para localização e mapeamento na IARA são GPS, odômetro, IMU e LiDAR.

O GPS, odômetro e a IMU são utilizados para inferir a posição do carro. Já o LiDAR (*Light Detection and Ranging*) é utilizado para se obter informações dos obstáculos no ambiente. O LIDAR funciona como um radar, porém com raios de luz, sendo capaz de emitir lasers em várias direções e gerar uma nuvem de pontos com a posição relativa dos obstáculos atingidos,(SCHWARZ, 2010). O sensor LIDAR da marca Velodyne, utilizado na IARA está representado na Figura 2.

No que tange o mapeamento, a IARA aplica uma solução em duas etapas, a primeira feita

Figura 1 – Carro autônomo IARA.

Fonte: Wiki do LCAD, acesso em 14 mai. 2018.

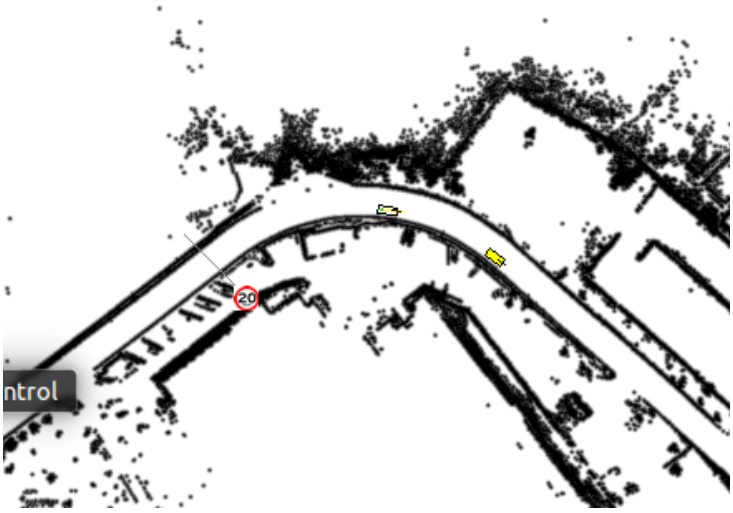
Figura 2 – Sensor LiDAR da marca Velodyne.

Fonte: Velodyne LiDAR, acesso em 14 mai. 2018.

offline e a segunda *online*. Na etapa *offline*, é aplicado uma técnica de SLAM (*Simultaneous Localization and Mapping*) para gerar as poses e o mapeamento, encarregado de gerar um mapa de ocupação de base da região onde a IARA irá trafegar. Para isso, é necessário primeiramente realizar um percurso de captação de dados dos sensores mencionados, com o veículo sendo dirigido por um motorista humano. Posteriormente, um algoritmo de GraphSlam (MUTZ et al., 2016), a partir da fusão de dados do GPS, Odometria, IMU e Lidar, é utilizado para estimar a posição do robô de forma que estas reflitam as poses mais precisas possíveis de onde o veículo passou no mundo real. (MUTZ et al., 2016)

Como essa fusão de dados é muito complexa, não é possível realizá-la em tempo real, durante a navegação autônoma. É por esse motivo, em adição ao fato de que a IARA é

Figura 3 – Mapa *offline* utilizado no simulador da IARA.



Fonte: Simulador da IARA, acesso em 25 mai. 2018.

baseada em localização precisa usando mapas, que essa etapa é chamada de *offline*.

Após a conclusão da etapa *offline*, o mapeamento *online* busca atualizar as informações do mapa de ocupação para incluir as modificações ocorridas como: veículos, pedestres, etc. Além disso a localização do veículo é feita comparando o mapa inferido em tempo real com o criado na etapa *offline*. A Figura 3 é um exemplo do mapa *offline* sendo utilizado pelo simulador do carro para navegação *online*.

A inferência de mapas a partir da nuvens de pontos do sensor laser é feita tanto na etapa *online* quanto na etapa *offline*. Como as leituras do LiDAR apresentam imprecisões, devido a condições ambientais (por exemplo chuva, folhas caindo, superfícies com reflexão irregular) ou erros de medida devido a ruídos do próprio sensor, não é possível confiar completamente nas distâncias e posições medidas por ele. Por isso, há a necessidade de se utilizar algum algoritmo que leve em consideração todas essas incertezas e procure filtrar ao máximo possível os erros de leitura do sensor.

O modelo utilizado atualmente na IARA busca atualizar a probabilidade de ocupação das regiões a cada nova leitura de ocupação do sensor LiDAR. Para isso, é aplicado um algoritmo de Occupancy Grid Mapping (OGM) (MUTZ et al., 2016) baseado no filtro de bayes que gradualmente aumenta a confiança de ocupação de uma célula. Contudo, essa filtragem temporal das inferências acaba não sendo muito precisa para objetos a longas distâncias. Além disso, é necessário a utilização de um algoritmo para inferir a ocupação de células que não foram atingidas por nenhum raio do sensor, já que o OGM utilizado não leva em consideração contextualidades espaciais.

Além disso, para que o mapeamento seja possível na IARA, é necessário o tratamento de vários casos especiais e milhares de linhas de código buscando a melhor forma de extrair as características da leitura do sensor. Em contrapartida, algoritmos de aprendizado de máquina tentam fazer essa inferência a partir de dados utilizados no treinamento.

Motivado por essas limitações do método atual de inferências de mapas, bem como pela potencialidade que a inteligência artificial tem mostrado em vários projetos da atualidade, esse trabalho propõe um novo modelo de inferências em tempo real de mapas de ocupação baseado em redes neurais. Os próximos capítulos serão dedicados a comprovar a capacidade desse método tanto pela revisão da literatura, quanto pela aplicação prática no mapeamento da IARA.

1.1 Motivação

A principal motivação deste trabalho é utilizar implementação em redes neurais para substituir o mapeamento de obstáculos devido ao seu potencial para reduzir centenas de linhas de código da IARA. Dessa forma o algoritmo OGM será substituído por uma descrição de rede neural convolucional com os seus pesos, que utilizará exemplos para aprender a tarefa. Espera-se então que a rede neural consiga aprender todas as nuances implementadas explicitamente a partir apenas dos exemplos de entrada e saída.

Algumas vantagens da utilização de algoritmos neurais, destacadas por Haykin e Engel (2007, p. 29), são:

- Não-linearidade: "Uma rede neural constituída por conexões de neurônios não-lineares é ela mesma não-linear";
- Mapeamento entrada-saída: "a rede aprende dos exemplos ao construir um mapeamento de entrada-saída. [...] não são feitas suposições prévias sobre o modelo estatístico dos dados de entrada";
- Adaptabilidade: "As redes neurais têm uma capacidade inata de adaptar seus pesos sinápticos a modificações do meio ambiente";
- Informação Contextual: "O conhecimento é representado pela própria estrutura e estado de ativação de uma rede neural. Cada neurônio da rede é potencialmente afetado pela atividade de todos os outros neurônios na rede. Consequentemente, a informação contextual é tratada naturalmente pela rede neural"

Trazendo a tona a problemática da geração de mapas de ocupação, as vantagens supracitadas mostram-se adequadas. A não-linearidade, por exemplo, aponta a capacidade de redes neurais para implementar funções que levam em consideração as complexidades inerentes da utilização de sensores submetidos ao ambiente (ruídos e condições adversas).

O fato de que este método é supervisionado, ou seja, gerado a partir de exemplos, também é desejável. Com a IARA, é possível extrair dados de entrada e saída para a rede a partir do próprio mapeamento já implementado no carro. Isto significa uma grande quantidade de dados anotados que serão utilizados para o treinamento.

Uma outra questão a se ressaltar é a característica da adaptabilidade, já que as ruas e vias por onde passam carros são ambientes extremamente variados. As ruas podem ser de terra ou asfalto, possuir ou não marcações, calçadas, etc.

A característica da contextualidade, por sua vez, mostra o potencial da rede para apresentar resultados ainda melhores que o mapeamento probabilístico. Isso por que o mapeamento bayesiano utilizado hoje, apesar de filtrar ruídos temporalmente nas células, não utiliza informações dos espaços vizinhos para estimar a ocupação. Porém, é razoável acreditar que em objetos comuns se todas as células em volta de uma central estão ocupadas, essa estará ocupada também. Essa consideração contextual está naturalmente embutida nas redes neurais, e podem inclusive ser favorecidas por algumas implementações da rede.

Para este aspecto da contextualidade, existe uma modalidade especial de rede neural, chamada de convolucional, que pelos resultados apresentados na literatura é promissora para o desafio de mapeamento proposto neste trabalho. Esse tipo de rede será explorada no projeto e tem seu funcionamento melhor explicado na seção Referencial Teórico.

Motivado por estas vantagens, bem como pelos ótimos resultados apresentados por redes neurais nas mais diversas áreas, como em processamento de imagens, de linguagem e sistemas de visão para carros, este projeto propõe a implementação neural e convolucional do mapa de ocupação.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo final deste trabalho é o de gerar o *occupancy grid map* para a IARA usando redes neurais a partir de dados de LiDAR. Para isso, o projeto foi dividido em cinco grandes áreas interdependentes. São elas: o estudo da literatura sobre mapeamento para robótica usando redes neurais; a escolha da arquitetura da rede; a preparação de bancos de dados; o treinamento da rede; análise dos resultados.

1.2.2 Objetivos específicos

- Estudo da literatura sobre mapeamento para robótica usando redes neurais;
- Escolha da arquitetura da rede;
- Preparação de bancos de dados;
- Treinamento da rede;
- Análise dos resultados do treinamento e teste da rede.

2 REFERENCIAL TEÓRICO

2.1 Introdução

Como esse trabalho propõe uma alternativa de mapeamento em grade ao método probabilístico atual da IARA, é importante que se entenda como funciona esse processamento de mapas anterior. Dessa forma, a seção 2.2 mostra primeiramente como inferências de mapas são integrados no processamento *offline* e posteriormente como é feita cada inferência de mapa no modelo probabilístico. Já a seção 2.3 explica o funcionamento de redes neurais, bem como mostra as aplicações que formam o estado da arte em processamentos neurais.

2.2 Localização e mapeamento

Em robótica móvel, o algoritmo de controle é responsável por fazer o robô "navegar"no ambiente até atingir uma posição específica em um dado sistema de coordenadas. Para isso, o robô precisa tomar uma decisão de movimento a cada instante, baseando-se em inferências de sua posição atual e do estado do ambiente à sua volta. Com isso, são derivadas duas áreas de estudo fundamentais: a localização e o mapeamento. Para estimar a localização se utiliza um conjunto de medidas extraídas de sensores, como velocidade e angulação. Esse par de medidas possui o nome de odometria do robô. Já a estrutura que agrupa localização e odometria se denomina *pose* do veículo (conjunto posição, velocidade e ângulo do volante). (THRUN; BURGARD; FOX, 2006)

O mapeamento constrói um modelo do ambiente em que o robô navega. Para realizá-lo, se utiliza sensores que agregam informações em relação à ocupação de espaços à sua volta. No caso de um mapa de ocupação para carros autônomos, o objetivo geralmente é descobrir se o espaço está ocupado de forma que possa gerar uma colisão.

Outra característica importante do mapeamento é que ele é representado por espaços discretos, por motivo de complexidade computacional. Ou seja, é impossível representar espaços contínuos no computador. Por esse motivo, um mapa de ocupação pode ser feito em grades (mapas em grade de ocupação, do inglês *occupancy grid map*). No entanto, quanto mais subdividido for o mapa, maior será o custo computacional da execução de cada iteração do mapeamento.

2.2.1 SLAM

Os domínios do mapeamento e localização são interdependentes, já que para obter uma localização mais precisa o robô precisa reconhecer padrões do ambiente. Por outro lado, para se identificar os padrões comparando com o conhecimento prévio do ambiente é necessário que se tenha noção da posição do robô em relação ao sistema de coordenadas. É preciso então estratégias para lidar com essa correlação para gerar simultaneamente uma localização e um mapeamento mais preciso.

O problema de se obter localização e mapeamento sem que nenhum dos dois seja previamente conhecido é chamado de SLAM. Alguns dos principais métodos para realizar o SLAM usam abordagens probabilísticas, buscando calcular a probabilidade de se estar em uma determinada posição x e com o mapa m no instante atual t, dado que as medidas dos sensores $z_{1:t}$ e comandos de controle $u_{1:t}$ foram feitas do instante inicial até o atual, como está representado na Equação (2.1).

$$P(x_{\mathrm{t}}, m \mid z_{1:\mathrm{t}}, u_{1:\mathrm{t}}) \tag{2.1}$$

- $\mathrm{x_t}$: é a posição do carro no instante t;
- m: é o mapa de ocupação;
- $\mathrm{z_{1:t}}$: são as medidas de sensores do instante inicial até o instante atual t;
- $\mathrm{u_{1:t}}$: são as os controles realizados pelo robô do instante inicial até o instante t.

Esse tipo de expressão é chamada de crença (tradução do inglês *belief*), e é resolvida iterativamente por alguma implementação do filtro de Bayes para duas variáveis aleatórias (x_t, m). Alguns exemplos modernos da implementação desse filtro para o SLAM, apresentados em Thrun, Bugard e Fox (2010, p. 337 e p. 437), são o GraphSlam e o fastSlam.

A implementação de SLAM atual da IARA é o Large-Scale Environment Mapping System. Essa é uma variação do GraphSlam que envia as nuvens de pontos do LIDAR e suas poses para um subsistema de mapeamento que otimiza as poses gerando uma localização precisa e aplica um OGM para gerar o mapa final. Essa etapa do mapeamento se chama de mapeamento *offline*. (MUTZ et al., 2016)

2.2.2 *Occupancy grid mapping*

O OGM se baseia na regra de Bayes, que quantifica a probabilidade condicional de uma variável aleatória em relação a outra, a partir da probabilidade inicial dessa variável e da probabilidade condicional inversa. Segue a expressão da regra para estimar a posterior de uma variável aleatória, retirada de Thrun, Bugard e Fox (2010, p. 17):

$$p(x \mid y) = \frac{p(y \mid x) * p(x)}{\sum_{x'} p(y \mid x') * p(x')} \tag{2.2}$$

Onde:

- *p(x / y)* é a probabilidade condicional de x dado y;
- *p(y / x)* é a probabilidade condicional de y dado x;
- *p(x)* é a probabilidade *a priori* de x;
- $\sum_{x'} p(y \mid x') * p(x')$ representa a soma de todos os estados possíveis.

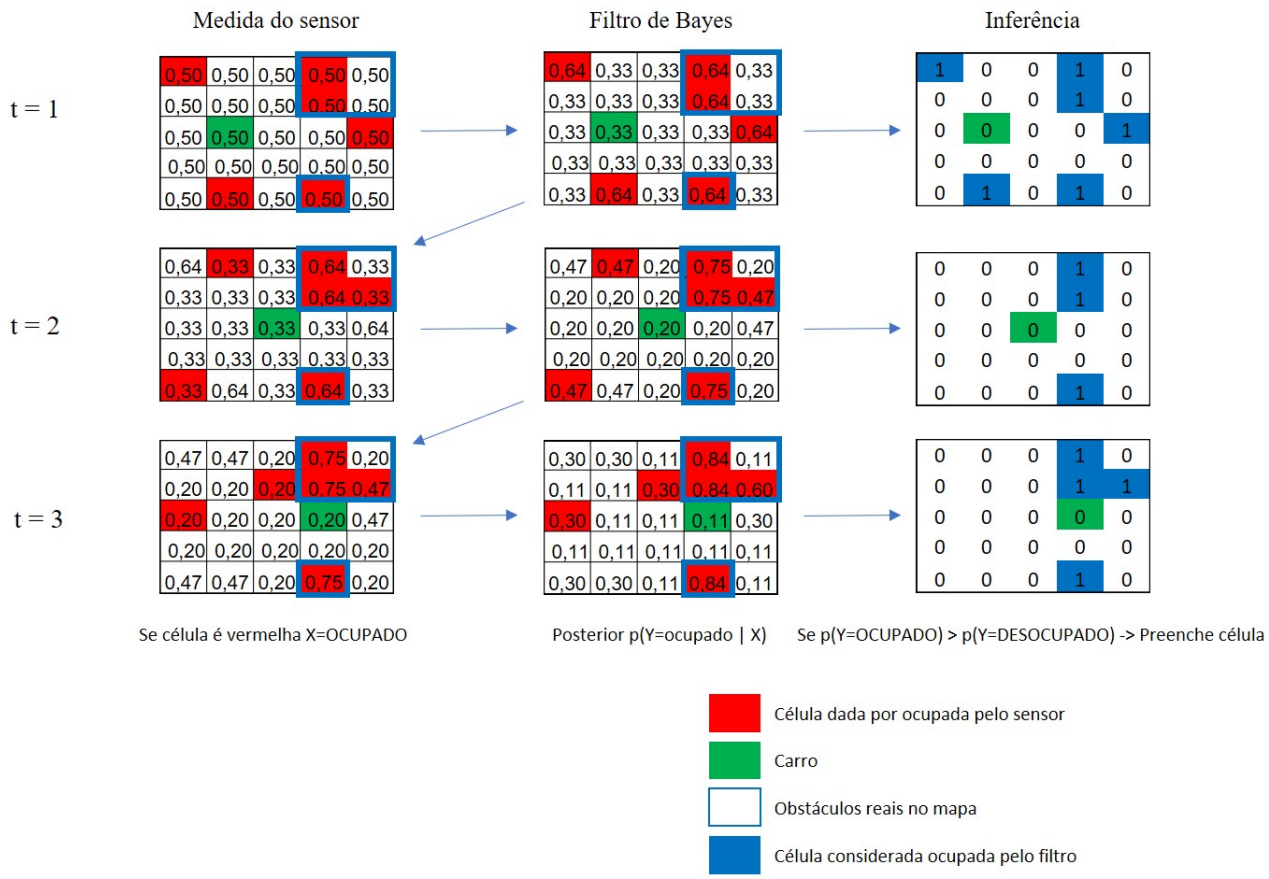
Para ilustrar esse processo aplicado a geração de um mapa de ocupação, bem como ressaltar possíveis falhas desse método, um exemplo inspirado em Pagac, Nebot e Durrant-Whyte (2007, p. 2) foi criado pelo autor e é representado na Figura 4.

Neste exemplo, a primeira coluna representa medições de ocupação (células vermelhas) em diferentes instantes de tempo, a segunda representa a saída do filtro de bayes e a última demonstra a inferência do mapeamento a partir destes dados. Como está representado na figura, primeiramente o sensor de Lidar faz um ciclo de medições, e a ocupação de uma região é inferida a partir da altura medida pelos lasers. Por exemplo, pode-se inferir que qualquer altura medida com mais de 50 centímetros dentro da região de uma célula conta como uma medida de célula ocupada do sensor.

Então, é feito o cálculo das probabilidades *p(Y=ocupado / X=inferência do sensor laser)* e *p(Y=desocupado / X=inferência do sensor laser)* a partir da regra de Bayes, a Equação 2.2. As células geralmente são inicializadas com uma probabilidade de 50% e existe um conhecimento prévio das probabilidades inversas (no caso, *p(X / Y)*).

A partir desse cálculo, é feita a inferência de ocupação da célula: se a probabilidade da célula estar ocupada é maior que a de estar desocupada, ela é marcada como ocupada e

Figura 4 – Exemplo de Mapeamento Bayesiano.



Fonte: Produção do próprio autor.

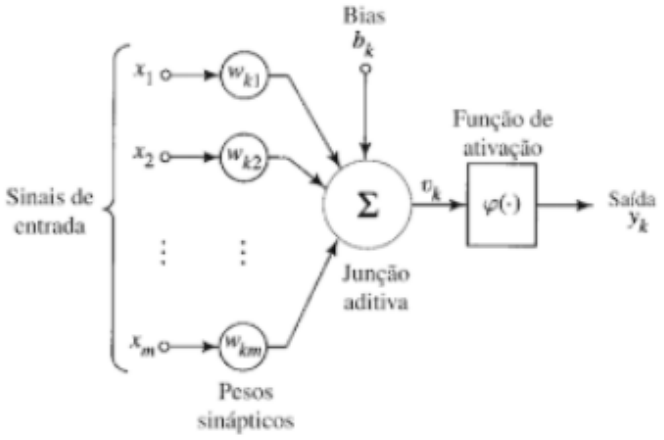
vice-versa. O leitor pode observar nesse exemplo que apesar de algumas medidas erradas do sensor(na primeira célula em t = 1 e na segunda célula em t = 2, por exemplo), o algoritmo consegue encontrar os contornos dos objetos em t = 3. Por outro lado, observa-se também que a célula superior mais à direita não foi dada como ocupada, apesar de efetivamente fazer parte do objeto dos três quadrados em volta dela.

Este fenômeno ocorreu pois nenhum laser conseguiu atingir a região, já que eles foram interrompidos pela parte mais próxima do objeto. Como as redes neurais aportam contextualidade entre os neurônios, espera-se que elas consigam identificar células pertencentes a objetos que não foram atingidos por raios de laser.

2.3 Redes neurais

Um tipo de algoritmo que tem sido amplamente utilizado como alternativa para solucionar desafios de natureza não determinística é o método das redes neurais artificiais. Esse

Figura 5 – Representação de um neurônio artificial.



Fonte: Haykin e Engel (2007, p. 36).

método, associado ao domínio da inteligência artificial, busca reproduzir a forma como o cérebro gera conhecimentos a partir dos sentidos.

Uma rede neural artificial é definida como um processador paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso. (HAYKIN; MARTINS, 2007)

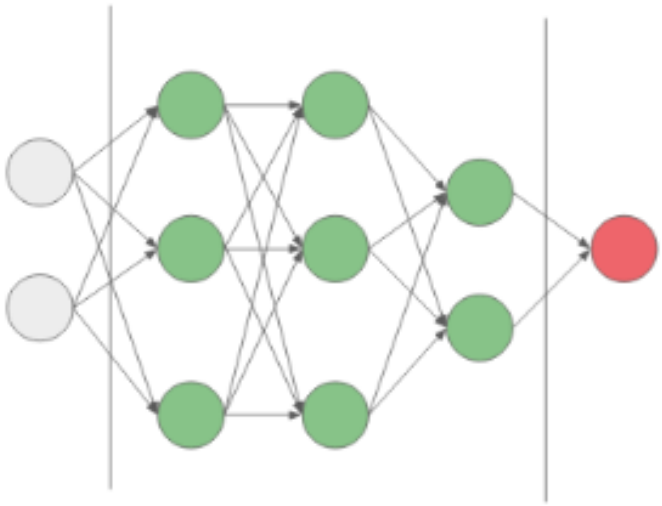
O modelo neural consiste, então, de uma rede de "unidades simples", os neurônios, que são capazes de processar dados de entrada em saídas que servirão de entrada para neurônios posteriores. A interligação entre os neurônios é chamada de sinapse. Cada sinapse entre dois neurônios possui um peso que deve ser considerado no processamento do neurônio, de maneira a ponderar as conexões.

Como se pode ver na Figura 5, a abstração do neurônio para computação faz uma soma ponderada por pesos das entradas e um valor constante (Bias). Essa soma é posteriormente aplicada a uma função de ativação que define a saída numérica do neurônio.

Assim, uma rede neural é determinada para um problema interligando neurônios às entradas, passando por camadas internas com seus pesos até chegar na camada final, onde se espera que a função resultante da rede reproduza o conhecimento esperado. Esse procedimento de propagação das entradas da rede, feita pelos neurônios, é comumente chamada de *feedforward*.

Em casos complexos é impossível a determinação por conta própria dos pesos que satisfazem a rede. Por este motivo, a rede passa por um processo de otimização para encontrar o

Figura 6 – Representação de rede neural.



Fonte: Produção do próprio autor.

conjunto ótimo de pesos. Parte desse processo é feito por um outro procedimento, chamado de *backpropagation*, cujo objetivo é treinar a rede para que chegue o mais próximo o possível de reproduzir as saídas esperadas a partir das entradas. Então, para realizar esse treinamento da rede, um conjunto de pares de entrada e saída sãos fornecidas pelo programador, para que a rede busque gerar saídas parecidas com as fornecidas a partir da otimização dos seus pesos. (HAYKIN; MARTINS, 2007)

Essas saídas se chamam *ground truth* do sistema, exatamente por serem determinadas como verdadeiras ao serem utilizadas como exemplo. Dessa forma, a rede neural consegue estimar funções que relacionam múltiplas variáveis de entrada e saída a partir de treinamentos feitos sobre exemplos.

2.3.1 Função de ativação

Como foi citado anteriormente, a função de ativação é a função que recebe as entradas em um neurônio, multiplicadas pelos seus pesos, e processa esses dados, propagando um novo valor. Essa função tem por objetivo adicionar um caráter não-linear ao processamento, de maneira a possibilitar à rede a representação de funções mais complexas. (JAIN; MAO; MOHIUDDIN, 1996)

A função de ativação mais comum quando se trata em rede neurais é a sigmóide, represen-

tada na Equação 2.3.

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \tag{2.3}$$

Existem diversas funções de ativação que cumprem o mesmo papel, mas que possuem diferentes propriedades, como a RELU(LECUN; BENGIO; G. HINTON, 2015), ELU (**ELU**), entre outros.

2.3.1.1 Função de ativação ELU

Uma função de ativação de especial interesse para este trabalho é a ELU, *Exponential Linear Units*. A Equação 2.4, representa essa função.

$$ELU(x, \alpha) = \begin{cases} x & \text{se } x \geq 0 \\ \alpha(e^x - 1) & \text{c.c.} \end{cases} \tag{2.4}$$

Essa função de ativação tem sido muito utilizada devido a sua capacidade de gerar aprendizados mais precisos e mais rapidamente do que outros métodos tradicionais. Essa melhora é atribuída ao fato de que a ELU apresenta uma média de ativações mais próxima de zero, ao mesmo tempo que satura ativações muito negativas. (CLEVERT; UNTERTHINER; HOCHREITER, 2015)

2.3.2 Função de custo

A função de custo de uma rede neural é a relação que avalia a magnitude do erro entre uma inferência da rede neural e o *ground truth*. Ela é utilizada a cada iteração do treinamento como ponto inicial do backpropagation. Consequentemente, os pesos da rede são atualizados de acordo com a influência estimada de cada neurônio para esse erro, que é comumente chamado de *loss*. Dessa forma, a escolha dessa métrica tem grande influência nos que se define como erro para uma rede neural. (HAYKIN; MARTINS, 2007)

Algumas métricas comumente utilizadas são a de erro quadrático, erro absoluto, erro exponencial e entropia cruzada.

2.3.2.1 Função de custo de entropia cruzada

A função de custo de entropia cruzada é utilizada em redes neurais pois possui propriedades que evitam a desaceleração do treinamento a medida em que o erro diminui, se comparada com a função de erro quadrático ou a exponencial, (ACADEMY, 2017). Ela é utilizada para problemas de classificação com entradas x n-dimensionais tal que x é da forma (C, d_1, d_2, ..., d_{n-1}) em que C representa o número de classes a serem classificadas e (d_1, d_2, ..., d_{n-1}) indica a posição de x nas (n-1) dimensões, como está descrita na Equação 2.5, retirada da documentação do *framework* para redes neurais pytorch, (TORCH CONTRIBUTORS, 2017). Nessa função, *classe* representa a classe correta a qual $x_{d_1, d_2, ..., d_{n-1}}$ pertence.

$$loss(x_{d_1, d_2,..., d_{n-1}}, classe) = -x_{d_1, d_2,..., d_{n-1}}[classe] + \log(\sum_{j=0}^{C-1} \exp(x_{d_1, d_2,..., d_{n-1}}[j])) \quad (2.5)$$

Outra opção apresentada nesse *framework* é a de designação de pesos diferentes para as classes, de maneira que a Equação 2.5, é adaptada para a Equação 2.6. Nesas equação *pesos* é um vetor de tamanho C. O valor contido em *pesos[classe]* vai penalizar o erro em proporções diferentes para as diferentes classes na classificação. Esse tipo de técnica é utilizada para bancos de dados desbalanceados, ou seja, que possuem número de dados diferentes em cada classe. (TORCH CONTRIBUTORS, 2017)

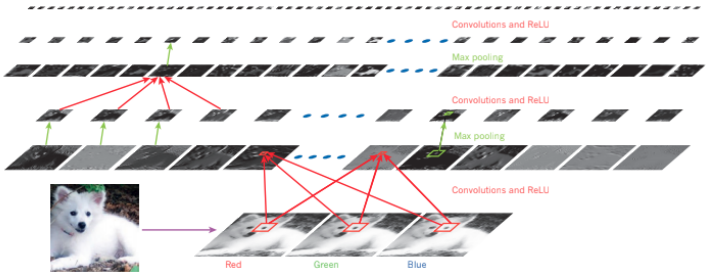
$$loss(x_{d_1,..., d_{n-1}}, classe) = pesos[classe] \times (-x_{d_1,..., d_{n-1}}[classe] + \log(\sum_{j=0}^{C-1} \exp(x_{d_1,..., d_{n-1}}[j]))) \quad (2.6)$$

2.3.3 Redes neurais convolucionais

Um tipo especial de rede neural para processar dados que vêm em listas multidimensionais, como imagens, é a rede neural convolucional. Neste tipo de rede, os pesos a serem treinados em uma camada são parâmetros de filtros convolucionais em paralelo. Os resultados dessas operações de convolução de uma camada com as entradas são chamados de *feature maps*. (SIMARD; STEINKRAUS; PLATT, 2003)

Esses filtros convolucionais, por sua vez, são matrizes a serem convoluídas com dados de entrada, também dispostos na forma de matriz. O operador de convolução é definido na

Figura 7 – Processo de filtragem de uma rede neural convolucional.



Fonte: Lecun, Bengio e Hinton (2015, p. 3).

Equação 2.7. Como se pode observar, o operador percorre os vetores, um na ordem direta e outro na ordem inversa, fazendo multiplicações entre seus valores. No caso de convoluções matriciais, o operador percorre a imagem de entrada tanto ao longo das linhas, quanto ao longo das colunas.

$$(F * k) = \sum_{s+t=p} F(s) \times k(t) \tag{2.7}$$

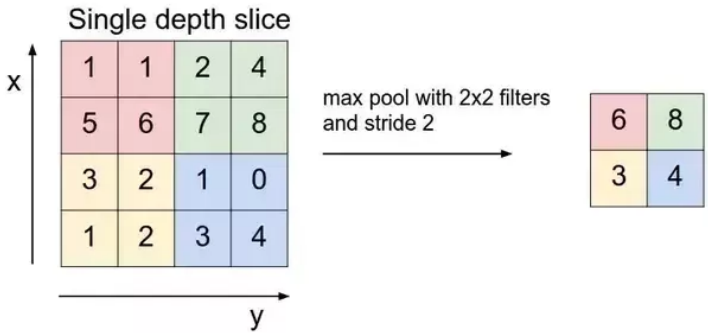
Filtros convolucionais são ferramentas muito utilizadas em processamento de imagens, sendo capazes de extrair contornos, normalizar histogramas entre outras coisas.

Em suma, considera-se que esse aspecto convolucional seja capaz de identificar padrões de correlação entre células próximas no desafio de mapeamento de ocupação com sensor LIDAR. De fato, desde o início dos anos 2000 as redes convolucionais têm sido aplicadas com grande sucesso em detecção, segmentação e reconhecimento de objetos. (LECUN; BENGIO; G. HINTON, 2015)

2.3.4 *Pooling*

Pooling é o nome que se dá a métodos de redução de amostragem de vetores com mínima perda de informação. Isso é feito ao se buscar invariância espacial na redução de dados, (SCHERER; MÜLLER; BEHNKE, 2010). *Max pooling* é o nome que se dá a redução de amostragem de matrizes feita a partir da extração do maior valor de cada vizinhança de tamanho pré-definido dentro da matriz, como está representado na Figura 8. Redes neurais utilizadas para processamento de matrizes geralmente utilizam o *max pooling* para reduzir o tamanho da matriz que será processada pela parte mais densa da rede. Dessa forma se reduz o custo computacional das operações com perda mínima de informação.

Figura 8 – Exemplo de max pooling.



Fonte: Convolutional Neural Networks (CNNs / ConvNets), acesso em 28 set. 2018.

É comum em redes neurais convolucionais se realizar um *pooling* para reduzir o custo computacional do processamento de grandes matrizes em camadas posteriores e no fim realizar um *unpooling* para retomar as dimensões originais. Esse *unpooling* deve decidir de que maneira mapear as informações de uma matriz com menos informações para uma matriz de saída maior. Não existe solução ótima para esse problema, já que não se pode definir uma operação inversa exata para a operação de *pooling* pois os valores não máximos são perdidos. A técnica que geralmente acompanha o *max pooling* é chamada de *max unpooling*, e consiste em guardar as posições das quais os maiores valores foram retirados na primeira operação, para depois computar uma inversa parcial da matriz que é composta dos valores da matriz de entrada do *unpooling* nas posições salvas, colocando zero nas outras posições.

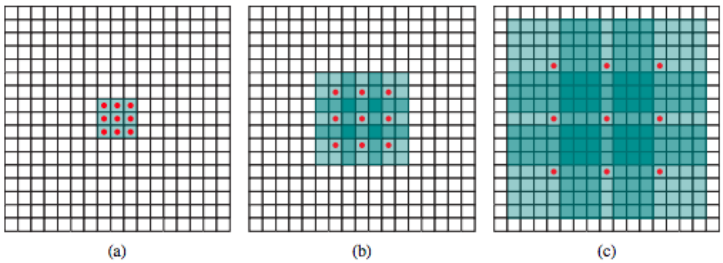
2.3.5 Camadas de convolução dilatada

Um tipo especial de convolução utilizada em redes neurais convolucionais é a convolução dilatada. Ela é feita a partir da mudança do operador de convolução discreta, de maneira a permitir um salto de um fator inteiro a cada operação. A Equação 2.7 representa a convolução discreta comum, enquanto a Equação 2.8 representa a convolução dilatada.

$$(F *_l k) = \sum_{s+(l*t)=p} F(s) * k(t) \tag{2.8}$$

Esse tipo de convolução é um operador que aplica o mesmo filtro convolutivo à distâncias diferentes usando fatores de dilatação diferentes. Dessa maneira, o campo receptivo, que é definido como a área de influência do filtro em uma matriz, é aumentado, sem perda de resolução ou cobertura, (YU; KOLTUN, 2015). Como se pode observar pela Figura 9, o

Figura 9 – Exemplo de dilatação do filtro de convolução.



Fonte: (YU; KOLTUN, 2015).

campo receptivo passa de (3x3) com dilatação 1 em (a), para (5x5) com dilatação 2 em (b) e (9x9) com dilatação 3 em (c).

2.3.6 *Dropout*

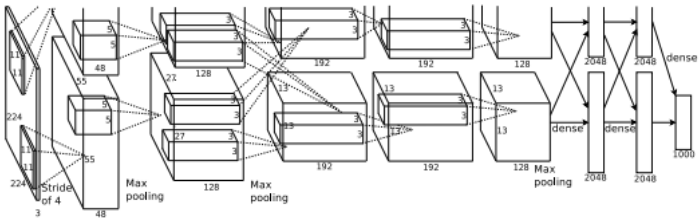
Dropout é uma técnica de regularização utilizada em redes neurais em que se retira alguns neurônios acultos aleatórios da rede durante o treinamento, de forma a a realizar o *feedforward* e o *backpropagation* em um subconjunto de dados sem eles. Esse procedimento é realizado pois tem a capacidade de melhorar a regularização do treino, como descrito em (WU; GU, 2015).

2.3.7 Estado da arte em redes neurais profundas

Um dos grandes avanços que permitiram o desenvolvimento na prática de redes neurais profundas foi a utilização da GPU para atualização e treinamento da rede. A partir dessa inovação e com o aumento da disponibilidade de dados, um dos primeiros trabalhos de classificação de imagens bem sucedido com redes neurais profundas é a AlexNet. Sua arquitetura liga diversas camadas convolucionais com camadas completamente conectadas e camadas de pooling, que buscam reduzir a quantidade de amostras da rede. Este é um modelo denso de rede, mas que possui alta taxa de acertos em classificações. (KRIZHEVSKY; SUTSKEVER; G. E. HINTON, 2012)

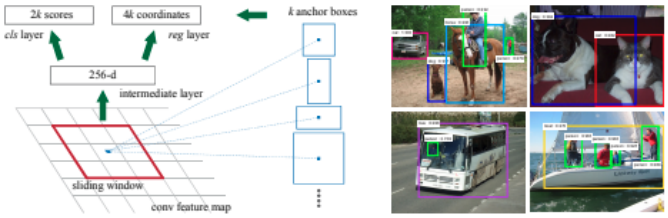
Outra rede neural profunda que é utilizada para classificação e localização de objetos em imagens é a R-CNN (*Faster Region-based Convolutional Neural Network*). Esta implementação busca primeiramente repartir a imagem em regiões de interesse para depois aplicar cada uma dessas regiões em uma rede convolucional. (REN et al., 2015)

Figura 10 – Arquitetura da AlexNet.



Fonte: Krizhevski, Sutskever e Hinton (2017).

Figura 11 – Funcionamento da faster r-cnn.



Fonte: Ren, He, Girshick et al. (2017).

Tabela 1 – Resultados comparativos do LoDNN na detecção de ruas.

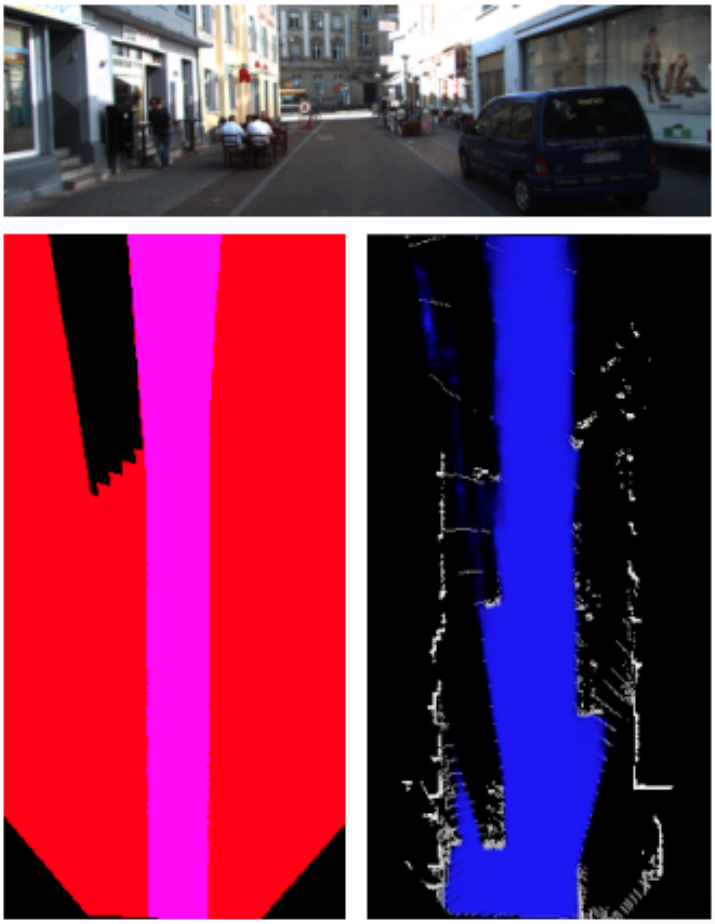
Método	MaxF	AP	PRE	REC	Tempo (ms)
LoDNN	**94,07**	**92,03**	92,81	**95,37**	**18**
Up-Conv-Poly	93,83	90,47	94,00	93,67	80
DNN	93,43	89,67	**95,09**	91,82	2000
FTP	91,61	90,96	91,04	92,20	280
FCN-LC	90,79	85,83	90,87	90,72	30
HIM	90,64	81,42	91,62	89,68	7000
NNP	89,68	86,50	89,67	89,68	5000
RES3D-Velo	86,58	78,34	82,63	90,92	360

Fonte: Caltagirone, Scheidegger e Svensson (2017, tradução livre).

Por último, citamos o modelo de rede completamente convolucional LoDNN (*LIDAR-based road detection neural network*).Este modelo foi realizado para identificar ruas e vias a partir de dados de LiDAR do banco de dados KITTI (GEIGER et al., 2013) e apresentou baixas taxas de erro nas inferências, bem como capacidade de execução em tempo real. (CALTAGIRONE et al., 2017)

Pode-se observar, na Tabela 1, os resultados desta rede em comparação com outros métodos de detecção de ruas no KITTI vision benchmark. O modelo apresentou um tempo de processamento baixo e altos índices percentuais em classificadores de acerto.

Figura 12 – Exemplo de entrada e saída do LoDNN.



Fonte: Caltagirone, Scheidegger e Svensson (2017).

2.4 Ferramentas e parâmetros de projeto

2.4.1 *Framework* de desenvolvimento

Existem inúmeros frameworks para implementação de redes neurais no mercado. Alguns deles são:

- Torch lua;
- Tensor Flow;
- PyTorch;
- Caffe;
- Keras.

O escolhido para este trabalho foi o PyTorch. Esse framework é uma versão do Torch Lua original, open source e implementado em Python (PASZKE et al., 2017). Devido a sua simplicidade de escrita aliada a flexibilidade para implementações, este framework possui uma grande comunidade ativa que contribui com inúmeras ferramentas de desenvolvimento.

Além disso, a versão atual do PyTorch conta com um front-end para C++ que é adequado para adaptação dos modelos da rede à IARA, já que a implementação da mesma é em C/C++. Dessa forma, com o Pytorch é possível treinar o modelo com a linguagem Python, adequada para a computação científica, e com facilidade se pode utilizar esse modelo posteriormente em um módulo do veículo.

2.4.2 Modelo da rede

A escolha do modelo de rede adequado passou pela identificação de trabalhos de natureza semelhante ao objeto de pesquisa.

Primeiramente, deve-se lembrar que a realização de um OGM se trata de inferir para cada célula do mapa a sua ocupação. Em outras palavras, tem-se duas classes possíveis para as células do mapa, ocupado e desocupado, e é necessário decidir a qual classe a célula pertence. Essa modalidade de classificação é chamada de segmentação semântica, geralmente aplicada à imagens.

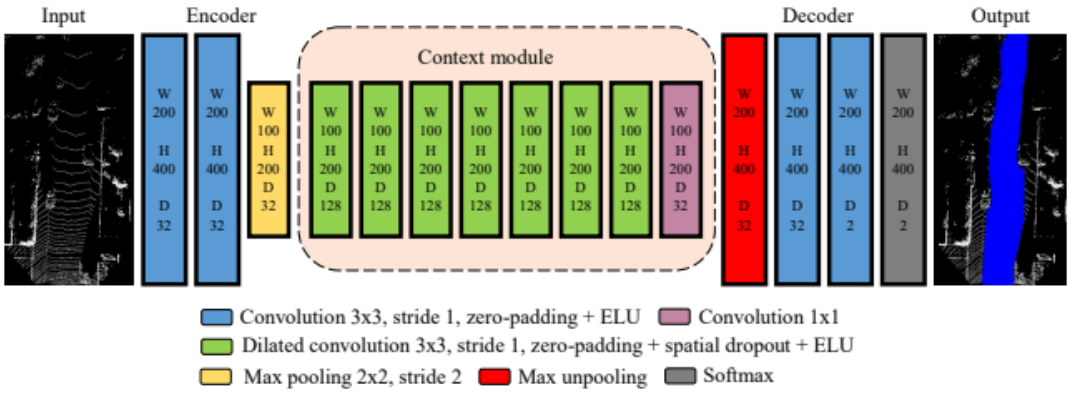
A seção 2.3.7 mostra alguns modelos de redes de segmentação semântica de código aberto. Dentre os modelos demonstrados, destaca-se o apresentado por Caltagirone, Scheidegger e Svensson (2017), já que a identificação de ruas a partir do sensor LiDAR proposta por eles é algo muito semelhante à identificação de obstáculos com o mesmo sensor, proposta neste trabalho.

Dessa forma, a arquitetura do modelo *LIDAR-based road detection neural network* (LoDNN) foi a escolhida para este trabalho. Portanto ela foi implementada em python, com o framework PyTorch. A Figura 13 mostra a arquitetura dessa rede neural, que é dividida em três grandes grupos de camadas, chamados de *Encoder*, *Context Module* e *Decoder*.

2.4.2.1 Pré-processamento da entrada da rede

A entrada da LoDNN é composta por cinco mapas de estatísticas em relação à ocupação de pontos em volta do carro. Esses mapas de estatísticas são representados por imagens

Figura 13 – Arquitetura da rede Fast LIDAR-based road detection.



Fonte: Caltagirone, Scheidegger e Svensson (2017).

de formato PNG, onde cada pixel representa um espaço de 20x20cm e o carro está sempre centralizado. Assim, cada posição atingida pelo laser irá computar a altura máxima, altura mínima, altura média, desvio padrão e número de pontos.

Para realizar esse pré-processamento, primeiramente os pontos da nuvem sofrem uma transformação de coordenadas, de esférica para cartesiana. Depois, a estatística da célula de posição correspondente às coordenadas apontadas pela leitura do LIDAR são atualizadas, gerando posteriormente as imagens que representam esses mapas de estatísticas da leitura do sensor. Por fim, essas cinco dimensões de estatísticas (altura máxima, mínima, desvio padrão e número de pontos) são colocadas como entrada na rede.

2.4.2.2 Encoder

O *Encoder* é a etapa encarregada de receber essa entrada da rede e reduzir a dimensão dos dados de entrada com uma perda mínima de informação. Para isso, são utilizadas duas camadas convolucionais normais, seguidas de uma camada de *max pooling*. (CALTAGIRONE et al., 2017)

2.4.2.3 Context Module

O *context module* está representado na Tabela 2. A principal ferramenta utilizada no *context module* da rede é a convolução dilatada.

Essas camadas apresentam um aumento progressivo da dilatação, de modo que na última

Tabela 2 – Descrição do Context Module da LoDNN.

Camada	1	2	3	4	5	6	7	8
Tamanho do filtro	3x3	3x3	3x3	3x3	3x3	3x3	3x3	1x1
Dilatação	(1,1)	(1,2)	(4,8)	(8x16)	(16,32)	(32x64)	(32x64)	-
Campo receptivo	3x3	5x7	9x15	17x31	33x63	65x127	129x255	129x255
Feature maps	128	128	128	128	128	128	128	32
Não-linearidade	ELU	ELU	ELU	ELU	ELU	ELU	ELU	-

Fonte: Caltagirone, Scheidegger e Svensson (2017, tradução livre).

camada exista uma dilatação de 32x64, ou seja, cada iteração do filtro vai considerar células a 32 posições de distância em um eixo e 64 posições no outro.

Diferentemente do trabalho proposto por Caltagirone, Scheidegger e Svensson (2017), neste trabalho foi determinado que o mapa de saída seria quadrado, com alcance de 60 metros em volta do carro. Portanto a dilatação utilizada em cada camada foi de mesma magnitude.

De acordo com o que foi discutido na Seção 2.3.5, as convoluções dilatadas possuem a capacidade de aumentar o campo receptivo do filtro convolucional, de maneira a aumentar a capacidade de detecção de contextualidade entre as células do filtro. Para o caso presente, de detecção de obstáculos, esse tipo de contextualidade em grandes distâncias é interessante para a previsão da continuação de obstáculos para locais mais distantes, onde o sensor já não é muito eficaz.

2.4.2.4 Decoder

O *Decoder* é última parte da rede, onde os dados processados na rede recuperam suas dimensões originais com um unpooling. A implementação do PyTorch para o unpooling utiliza o método discutido na Seção 2.3.4, na qual as posições dos valores máximos extraídos no max pooling são guardadas, de modo a inserir os valores na entrada do unpooling nessas mesmas posições e igualar os outros à zero. (TORCH CONTRIBUTORS, 2017)

O decoder segue com mais duas camadas convolucionais normais. Por fim, os feature maps entram em uma camada de log softmax, que retorna uma saída de mapa multidimensional, em que cada dimensão é o logaritmo de probabilidade para cada classe.

2.4.2.5 Saída da rede

Com o intuito de facilitar a adaptação do modelo neural à IARA, foi decidido que a rede deverá retornar um mapa de probabilidades para três classes:

- Ocupado;
- Desocupado;
- Desconhecido.

Assim, o módulo de rede neural faz inferências da mesma maneira que o método probabilístico de inferências, possibilitando maior compatibilidade com outros módulos do carro. Portanto, a inferência de ocupação é feita a partir da comparação entre as probabilidades de cada uma dessas três classes na saída da rede.

3 METODOLOGIA EXPERIMENTAL

3.1 Visão geral do módulo de mapeamento

O módulo de mapeamento neural é um sistema que recebe como entrada os dados de laser da IARA e gera como saída o mapa de ocupação do entorno do carro. Essa sequência está ilustrada na Figura 14. Como se pode observar, é esperado que as inferências do mapeamento neural possam ser utilizadas inclusive para o mapeamento online da IARA, de maneira que ele deve ser rápido o bastante para funcionar em tempo real.

3.2 Geração do banco de dados

O treinamento da rede precisa de um banco de dados com exemplos de entrada e saída. Para criar esse banco de dados, o *ground truth* poderia ter sido anotado manualmente ou ter sido retirado de uma planta baixa do local. Entretanto, como o mapeamento *offline* da IARA representa uma fonte confiável e abundante de anotações, ela foi utilizada gerando pares de entrada e saída a partir de dados salvos de trajetos antigos do carro(*logs*). Com o simulador da IARA as nuvens do LIDAR e os mapas de ocupação do mapeamento offline foram extraídos e salvos na forma de imagens de formato PNG.

O log utilizado para treinamento neste trabalho foi o de uma volta no anel viário da UFES de madrugada, realizada em 07/09/2018. O anel viário está destacado na foto capturada

Figura 14 – Fluxograma de funcionamento do mapeamento neural.

Fonte: Produção do próprio autor.

Figura 15 – UFES vista de cima, com anel viário destacado.

Fonte: Foto de satélite retirada do Google Maps.

por satélite da UFES, na Figura 15.

Houveram dois motivos principais para a escolha desse log. Em primeiro lugar, destaca-se que essa volta foi escolhida por que os obstáculos presentes na UFES seriam conhecidos, o que facilitaria análises posteriores do mapeamento. Em segundo lugar, o fato de que essa volta foi realizada de madrugada reduziria a quantidade de imprecisões no *ground truth*, já que nesse horário a presença de objetos móveis é menor.

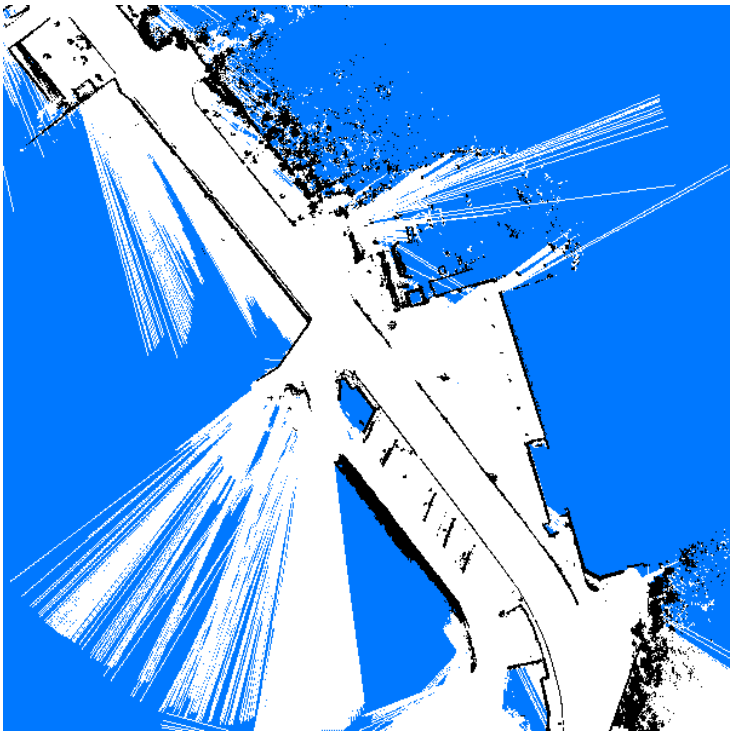
Outra informação importante é que os pares de entrada e saída foram gerados a cada dois metros percorridos pelo veículo. Dessa maneira, o banco de dados não tinha mais informações em regiões em que o carro andou mais lentamente no *log* simulado. Isso foi feito com o objetivo de se evitar que a rede tivesse um aprendizado desbalanceado para regiões diferentes, ou seja, para evitar que houvesse o overfit em determinada região.

3.2.1 Geração do Ground truth

O *ground truth* utilizado foi a saída do mapeamento offline, que acumula as nuvens do LIDAR com a integração de dados da odometria do veículo. Esse mapa foi escolhido ao invés do OGM pois ele é mais preciso devido à integração com a odometria e ao acúmulo de vários mapas instantâneos. Ainda, ele apresenta informações de ocupação mais distantes, o que reforça a possibilidade de uma rede neural que faz previsões de lugares com poucos pontos do laser. Assim, o *ground truth* é gerado a partir da tradução das probabilidades de ocupação do mapeamento offline para as classes de segmentação da rede (célula desconhecida, desocupada ou ocupada).

Por fim esse mapa de ocupação é exportado como imagem PNG a medida que o simulador

Figura 16 – Exemplo de Ground Truth da rede neural.



Fonte: Simulador da IARA.

disponibiliza o mapa de ocupação de determinado instante. A Figura 16 mostra um exemplo de mapa de ocupação utilizado como ground truth da rede.

3.2.2 Geração dos dados de entrada

Os dados de entrada da rede são cinco mapas de estatísticas em relação à medição do LIDAR no entorno do carro, como foi explicado na Seção 2.4.2.1. Dessa forma, foi necessário atualizar esses mapas a medida em que o simulador apresentava as leituras do sensor.

Ao se identificar a posição nos eixos x e y onde um raio do laser foi refletido, as estatísticas da célula que representa essa posição são atualizadas com a altura z a partir do seguinte algoritmo:

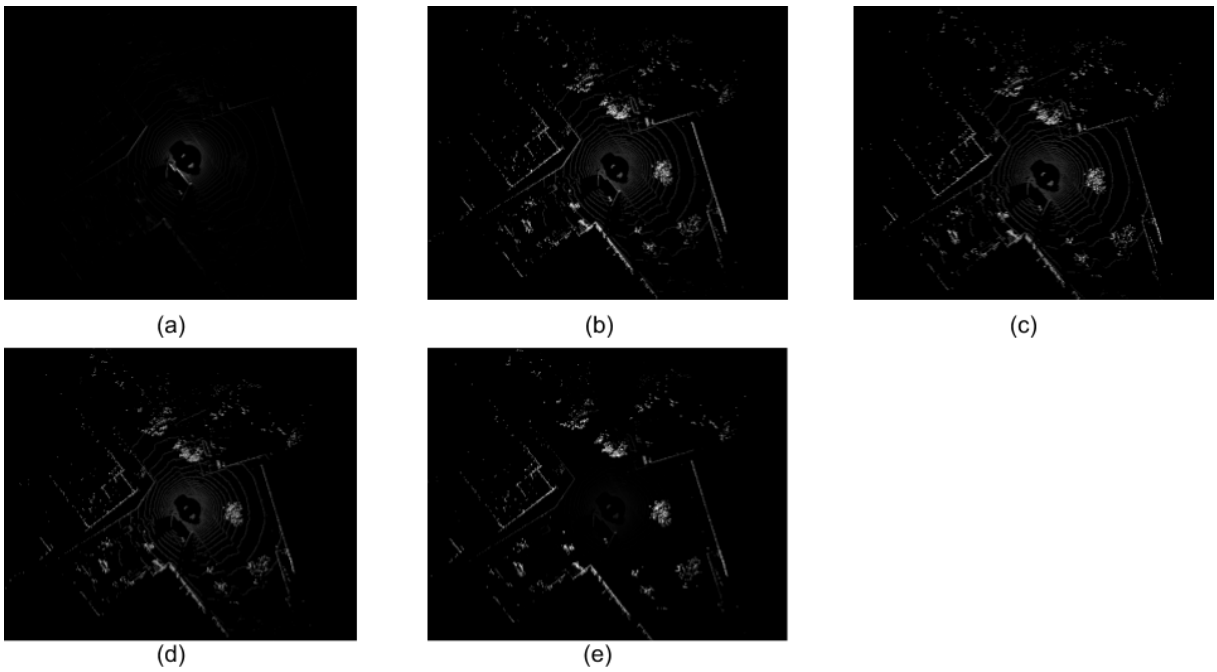
- Mapa de altura número de pontos:

$$n_{(x,y)} = n_{(x,y)} + 1$$

- Mapa de altura máxima:

$$max_{(x,y)} = z$$

Figura 17 – Cinco mapas de estatísticas processados no simulador da IARA. Na ordem, temos: (a) mapa de número de pontos; (b) mapa de altura máxima; (c) mapa de altura mínima; (d) mapa de altura média; (e) mapa de desvio padrão.



Fonte: Produção do próprio autor.

se

$$z > max_{(x,y)}$$

- Mapa de altura mínima:

$$min_{(x,y)} = z$$

se

$$z < min_{(x,y)}$$

- Mapa de altura média:

$$media_{(x,y)} = \frac{media_{(x,y)} \times (n_{(x,y)} - 1) + z}{n_{(x,y)}}$$

- Mapa de desvio padrão:

$$\sigma_{(x,y)} = \frac{\sum z^2}{n_{(x,y)}} - media^2_{(x,y)}$$

Ao fim de uma volta completa do sensor LIDAR os cinco mapas de estatísticas são salvos em conjunto com o *ground truth* no formato PNG. Foi certificado que os índices de entrada e a saída estavam na mesma posição. A Figura 17 mostra essas estatísticas de entrada.

3.2.3 Propostas de bancos de dados

A partir dos métodos apresentados, duas propostas de bancos de dados foram geradas a fim de que seus resultados fossem comparados.

São elas:

- Banco de dados sem acúmulo de nuvens;
- Banco de dados com acúmulo de 5 nuvens;

No banco de dados sem acúmulo de nuvens cada índice representa apenas uma leitura do LIDAR, enquanto no banco de dados com acúmulo cada índice agrega informações de cinco nuvens anteriores consecutivas. O segundo banco de dados foi gerado para a possibilidade de apenas uma volta do sensor não aportar informações suficientes em relação ao mapa *offline*, já que este é gerado a partir de informações de várias inferências consecutivas.

A Figura 18 mostra a diferença entre mapas de número de pontos acumulado e não acumulado, onde ambas as figuras estão normalizadas em escala de cinza para visualização. É evidente que o mapa acumulado aporta mais informações do que o não acumulado, então é de se esperar que ele apresente previsões mais precisas. Entretanto o acúmulo de mapas também aporta uma maior quantidade de operações no processamento do dados de entrada, então espera-se também um aumento no tempo total da operação.

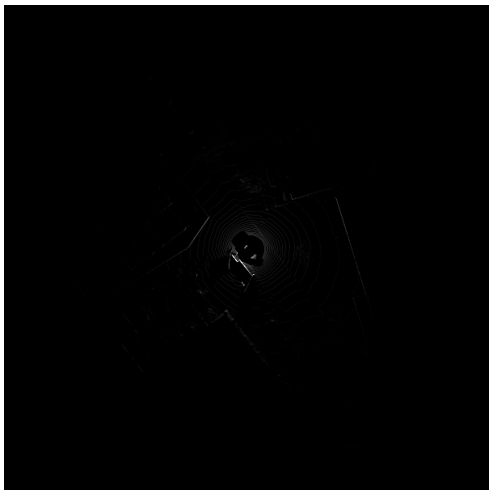
3.3 Treinamento

3.3.1 Parâmetros de treino

Foram utilizados os seguintes parâmetros para o treino:

- Dropout;
- Função de ativação ELU;
- Função de custo de entropia cruzada.

Figura 18 – Comparação entre dado de entrada acumulado e não acumulado.



(a) Dado de entrada não acumulado normalizado para escala de cinza.

(b) Dado de entrada acumulado normalizado para escala de cinza.

Fonte: Produção do próprio autor.

Como foi discutido na seção 2.3.6, o *dropout* tem o objetivo de reduzir o overfit no treinamento. Neste modelo, o dropout foi utilizado intercalando camadas comuns com camadas de dropout. Seguindo o trabalho apresentado por Caltagirone, Scheidegger e Svensson (2017), foi utilizado um dropout de 20%.

A função de custo de entropia cruzada e função de ativação ELU, explicadas em 2.3.1 e 2.3.2, também foram escolhidas de acordo com o trabalho em questão.

Figura 19 – Divisão do banco de dados entre train set e validation set.

Fonte: Foto de satélite retirada do Google Maps.

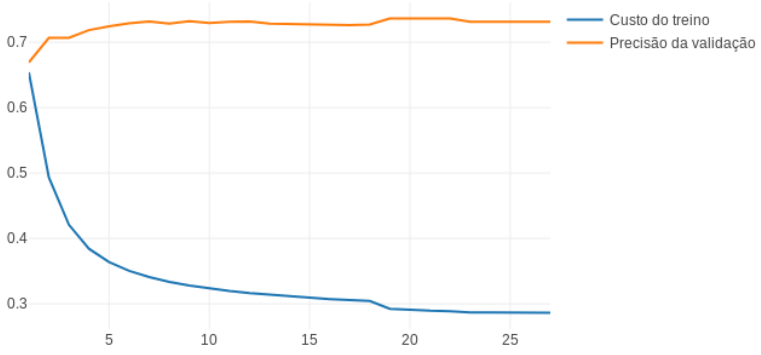
3.3.2 Divisão do banco de dados

Para validar o funcionamento de uma rede neural, o banco de dados em três parcelas, chamadas de treino, validação e teste. O conjunto de treino é o que será usado no processo de otimização da rede. Já o conjunto de validação é utilizado ao longo do treinamento para se verificar se a rede neural está conseguindo generalizar seu aprendizado, ou se apenas está gerando um overfit em relação aos dados de treino. Por fim, o conjunto de teste é usado para verificar os resultados após todo o treinamento. Assim, é considerado razoável uma divisão de 60% à 70% para treino e de 15% à 20% para validação e teste, a depender do tamanho do seu banco de dados.

Ainda, é muito importante que os bancos de dados de validação e teste não contenham informações duplicadas do banco de dados de treino, já que dessa forma geraria interpretações superestimadas em relação aos resultados do treinamento.

Como a volta da UFES representa um espaço limitado, incapaz de gerar um banco de dados grande, foi decidido que o banco de dados seria dividido apenas entre treino e validação, enquanto o teste final ficou a cargo da utilização da rede em outra volta mais recente do carro. Assim, a Figura 19 mostra a divisão que foi feita entre banco de dados de treino (vermelho) e banco de dados de validação (laranja). Foram escolhidos dois trechos distintos para validação para garantir a capacidade de generalização da rede.

Figura 20 – Dados de treino.



Fonte: Produção do próprio autor.

3.3.3 Procedimento de treino

Como foi dito em 2.4.1, o procedimento de treino foi implementado no PyTorch e consistiu basicamente de um loop que treinava ao longo de todo o banco de dados de treino e depois testava o modelo com os pesos atuais no banco de dados de teste, disponibilizando ao longo desse procedimento a *loss* de treino e teste, além da quantidade de pixels que o teste acertou em relação ao *ground truth* (acurácia). É definido que uma época de treino se passa quando se realiza o backpropagation em todo o banco de dados de treino uma vez. Assim, uma validação era realizada a cada época.

O método utilizado para avaliar os treinamentos era a partir da análise gráfica da redução da *loss* de treino e de validação, da seguinte maneira: enquanto a ambas as *loss* diminuem, o treinamento está funcionando e não está gerando overfit, já que a *loss* do treino também está reduzindo; se passarem algumas época em que apenas o treinamento teve redução de *loss*, possivelmente a rede está fazendo um overfit.

No Gráfico 20, por exemplo, pode-se observar que o treinamento com o banco de dados não acumulado chegou à um limite de precisão na época 18, já que a partir das próximas décadas a precisão do set de validação não aumenta mais, mesmo que a loss do treino diminua.

3.3.4 Otimizações do treino

Existem algumas práticas relatadas na literatura que podem ser utilizadas com o objetivo de se alcançar melhores resultados no treinamento de uma rede neural. Uma dessas práticas é o embaralhamento do banco de dados durante o treino. Ela foi utilizada para tornar o treinamento mais rápido, visto que a disposição dos batchs na ordem em que os dados foram extraídos do simulador gerava grupos com maior dificuldade de redução de *loss* que outros. Observou-se que com os dados embaralhados o treino fica mais uniforme, já que os batchs são formados por regiões diferentes. Assim, o banco de dados era primeiramente separado entre as regiões de teste e de treino, e depois era embaralhado no início do treino.

Como se pode observar na Figura 16, a quantidade de pixels da classe ocupado em cada imagem é muito menor do que as classes desocupado e desconhecido. Esse fenômeno gera uma tendência natural da rede para optar pelas duas últimas classes sobre a primeira. Para evitar isso, pesos inversamente proporcionais à quantidade de pixels das classes em cada batch foram utilizados, de forma que a loss function multiplica o peso no momento de computar o erro. Essa "valorização"dos erros em relação às classes menos frequentes compensa o desbalanceamento entre elas. (SIMARD; STEINKRAUS; PLATT, 2003)

Finalmente, foi realizado um aumento artificial do banco de dados, visto que os dados extraídos da volta da UFES geravam apenas 1600 pares de entrada e saída. As principais práticas utilizadas para aumento de bancos de dados de imagens são a rotação, o espelhamento e a translação, (PEREZ; WANG, 2017). No caso deste trabalho, o aumento do banco de dados foi realizado a partir da rotação de 15 em 15 graus de cada imagem de treino, aumentando assim em 24 vezes esse banco de dados.

3.4 Métrica de avaliação

A métrica escolhida para avaliação dos modelos treinados foi a de acurácia média das classes nas previsões da rede. Nesse método a classe prevista em cada célula na saída da rede é comparada com o *ground truth*. Dessa maneira, essa métrica está definida na Equação 3.1, onde VP é o total de células cujas classes foram previstas corretamente e TOTAL é o número total de células no banco de dados, VP e TOTAL consideram todas as imagens do banco de dados.

$$A = \frac{VP}{TOTAL} \tag{3.1}$$

Outra ferramenta utilizada para análise de resultados nos bancos de dados de teste foi a

Tabela 3 – Exemplo de matriz de confusão.

		Inferência da rede		
		Desconhecido	Desocupado	Ocupado
Ground Truth	Desconhecido	75%	10%	15%
	Desocupado	20%	60%	20%
	Ocupado	15%	10%	75%

Fonte: Produção do próprio autor.

matriz de confusão. Esse é um tipo de tabela que permite a visualização do desempenho de um algoritmo de aprendizado, geralmente um algoritmo supervisionado. Nela, cada linha representa as classes previstas pelo *ground truth*, enquanto as colunas representam as inferências da rede. Dessa forma, é possível com uma rápida visualização reconhecer que previsões geram mais erros. No caso desse trabalho, os valores da matriz de confusão são todos percentuais do total de cada classe. Na Tabela 3, por exemplo, pode-se observar que todas as linhas somam 100% no total, e ainda que a diagonal da matriz representa os acertos da rede em relação ao *ground truth*. Como exemplo de análise, temos que a rede é menos precisa na classificação da classe desocupado, já que acertou apenas 60%, enquanto nas outras classes a rede acertou 75%.

4 RESULTADOS

Como foi descrito na metodologia, a rede neural utilizada foi treinada com dois modelos de bancos de dados. São eles:

- Banco de dados instantâneo;
- Banco de dados acumulado.

4.1 Treinamento da rede

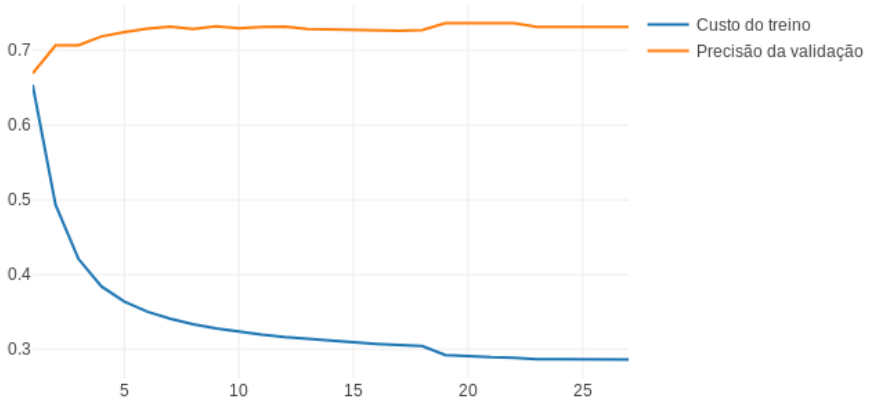
A primeira análise consiste na caracterização do aprendizado da rede com os dois bancos de dados. O treinamento foi mantido enquanto enquanto houvesse melhora significativa na loss do treino e nos acertos no banco de validação. Foram aplicadas reduções periódicas da taxa de aprendizado para alcançar resultados mais precisos.

A Figura 21 mostra a evolução do erro no banco de dados de treino e do acerto percentual no set de validação para o treinamento com dataset instantâneo. O eixo x mostra as épocas de treinamento passadas, enquanto o eixo y mostra o erro pela métrica de entropia cruzada no banco de dados de treino e a acurácia medida no banco de dados de validação. A taxa de aprendizado nesse treinamento começou em 0,0005, mudou para 0,0001 na época 17 e depois para 0,00005 na época 23, a partir da qual não houve mais quedas significativas na loss do treino. Como se pode observar, a rede alcançou uma acurácia de 73% na época 18, a partir da qual a queda na loss do treino não aportou melhoras no acerto do set de validação.

Agora em relação ao treino do banco de dados acumulado, temos a Figura 22. Observa-se pela evolução do treino que houve uma acurácia no banco de validação de 73% a partir da época 15, com uma *loss* no treino de 0,28. A taxa de aprendizado inicial foi de 0,0005, passando por 0,0001 na época 15 e terminou em 0,00005 a partir da época 25.

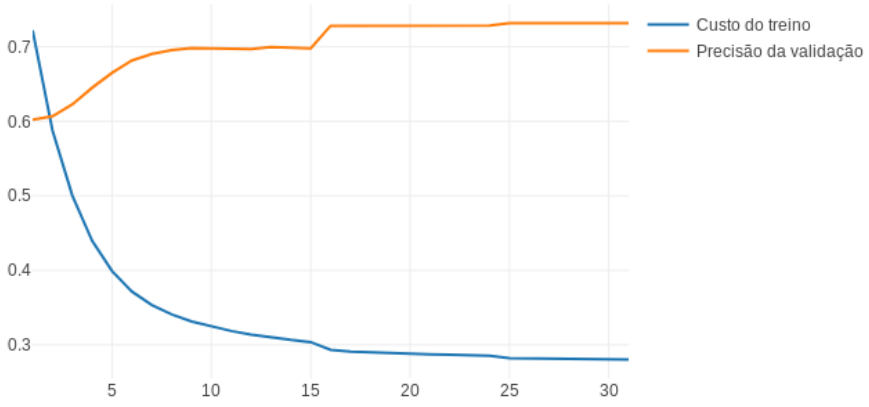
Comparando os dois resultados, concluiu-se que o acúmulo de nuvens para geração do banco de dados não resultou em uma queda na *loss* do treino (ambos acabaram com uma *loss* próxima de 0,28) e nem no aumento da acurácia na validação. De fato, se compararmos os exemplos de instantes parecidos na Figura 23 do banco de dados (a) não acumulado e (b) acumulado, não há uma melhora evidente na rede treinada com acumulação de nuvens do LiDAR.

Figura 21 – Evolução do treinamento para dataset instantâneo.



Fonte: Produção do próprio autor.

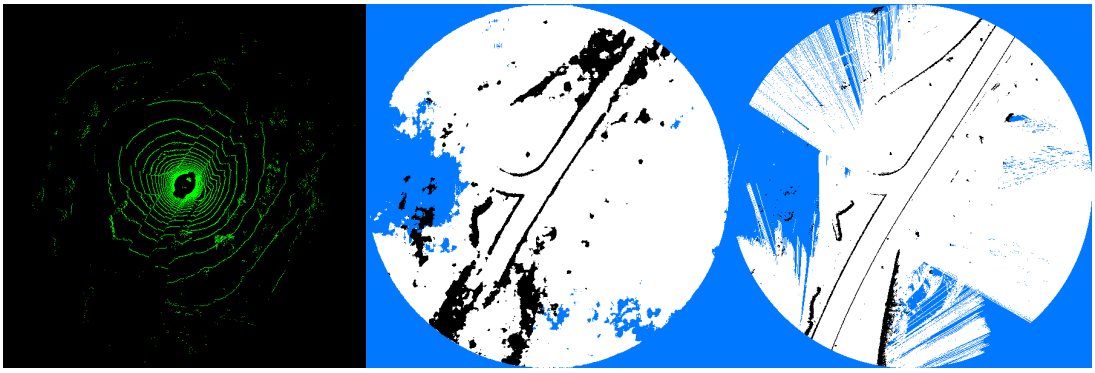
Figura 22 – Evolução do treinamento para dataset acumulado.



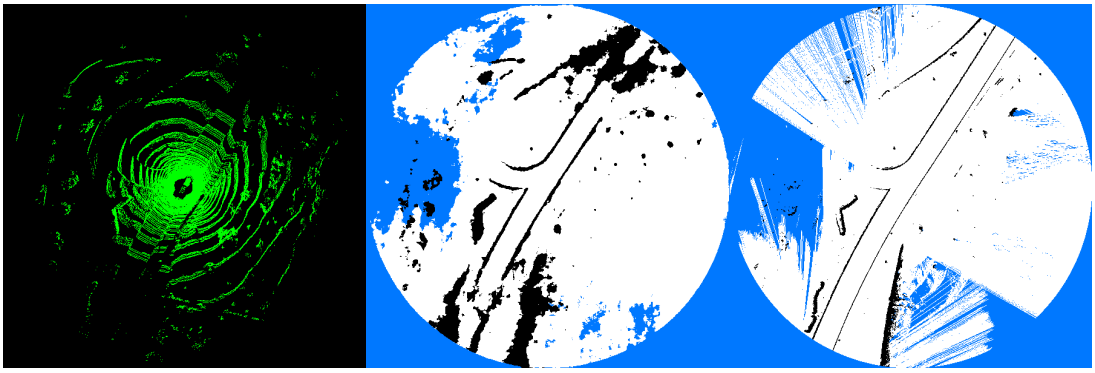
Fonte: Produção do próprio autor.

Figura 23 – Comparação entre predições dos modelos de rede com e sem acúmulo de nuvens do LIDAR.

(a) Entrada, predição e *ground truth* do modelo sem acúmulo de nuvens.



(b) Entrada, predição e *ground truth* do modelo com acúmulo de nuvens.



Fonte: Produção do próprio autor.

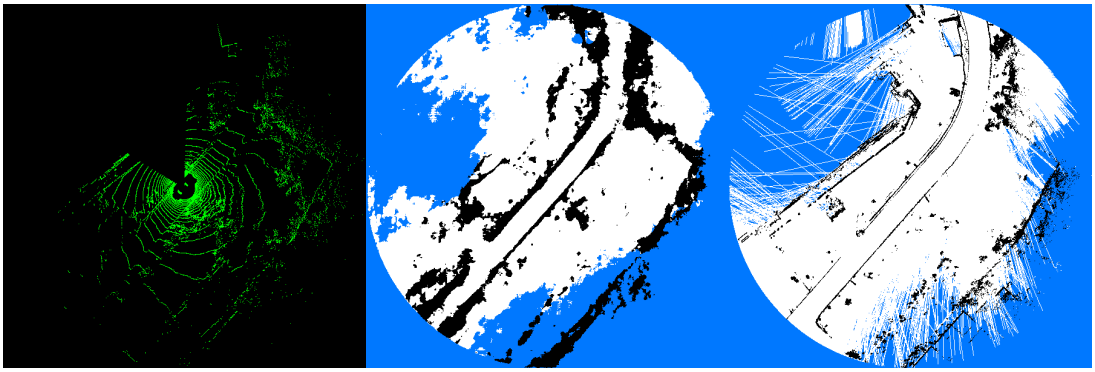
4.2 Teste da rede

Para validar a capacidade de generalização da rede de mapeamento, foram realizados testes em dois *logs* diferentes dos utilizados para treinamento. O primeiro foi um *log* de 23/03/2016 da volta da UFES. Como esse log foi gerado a dois anos atrás ele contém informações e contornos diferentes dos apresentados no log utilizado no treino de 2018. O segundo log de testes foi o de uma volta realizada no estacionamento do Centro de Pesquisas da Petrobras, CENPES, feito no dia 25/11/2018. O mapa foi feito da região demonstrada pela Figura 25.

A Figura 24, mostra os dados de entrada, predição da rede e *ground truth* do modelo (a) com banco de dados não acumulado e (b) com banco de dados acumulado para a volta da UFES em 2016. Esse banco de dados possui 400 pares de entrada e saída para teste. Já a Figura 26 mostra as mesmas imagens em um instante do dataset do CENPES, que possui 126 instantes salvos para teste. Estas imagens demonstram que a rede conseguiu generalizar seus resultados, já que foi capaz de identificar contornos em logs muito diferentes daquele utilizado para treino e validação.

Figura 24 – Teste da rede neural com dataset da UFES de 2016.

(a) Entrada, predição e *ground truth* do modelo sem acúmulo de nuvens.



(b) Entrada, predição e *ground truth* do modelo com acúmulo de nuvens.

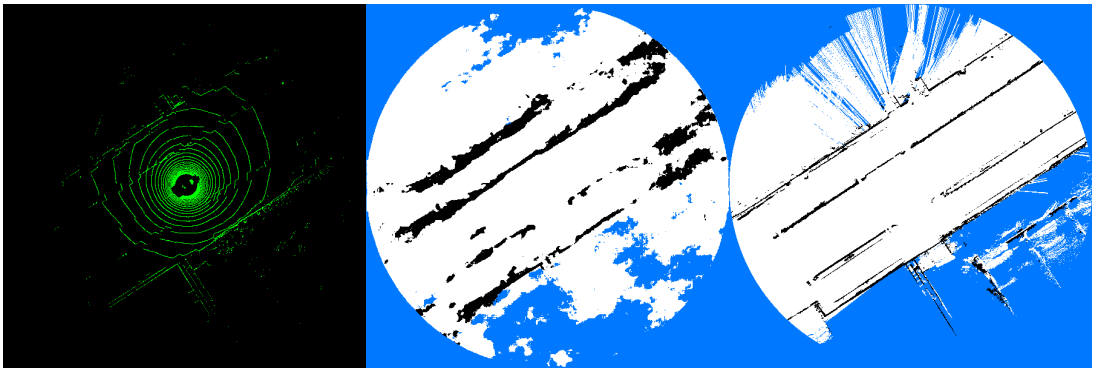
Fonte: Produção do próprio autor.

Figura 25 – Imagem de Satélite CENPES de 2018.

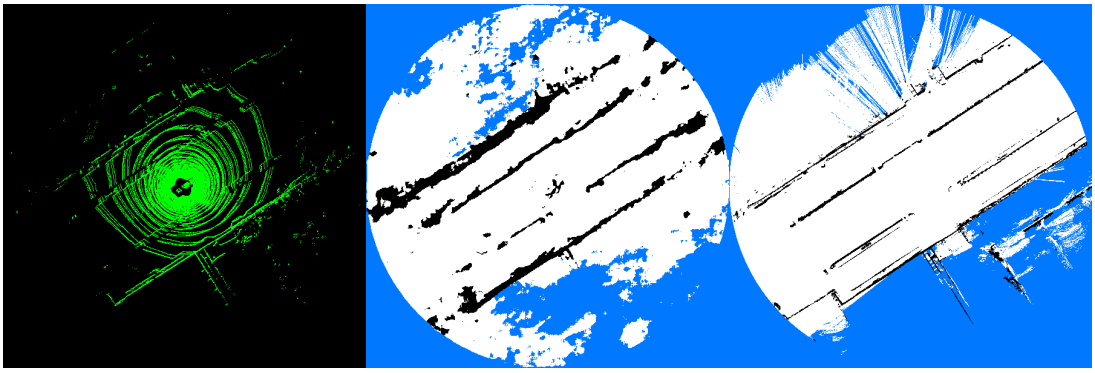
Fonte: Imagem de Satélite do Google Maps.

Figura 26 – Teste da rede neural com dataset da CENPES de 2018.

(a) Entrada, predição e *ground truth* do modelo sem acúmulo de nuvens.

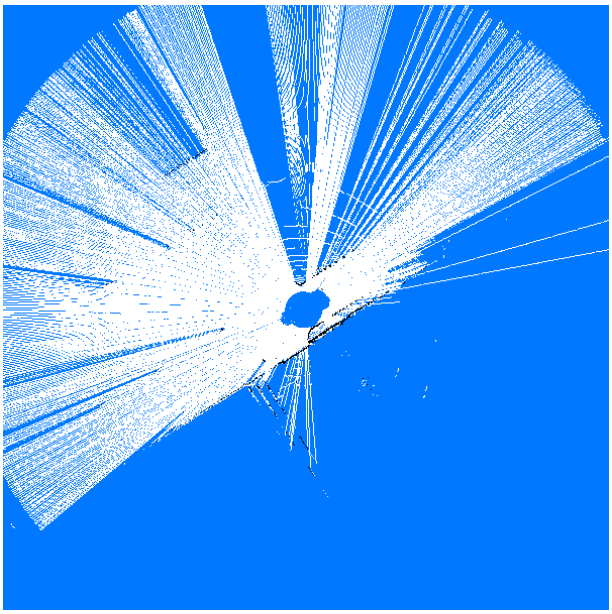


(b) Entrada, predição e *ground truth* do modelo com acúmulo de nuvens.



Fonte: Produção do próprio autor.

Figura 27 – Snapshot map com dataset da CENPES de 2018.



Fonte: Produção do próprio autor.

Tabela 4 – Matriz de confusão para o banco de dados não acumulado no *log* Volta da UFES de 2016.

		Inferência da rede		
		Desconhecido	Desocupado	Ocupado
Ground Truth	Desconhecido	48263585 (87%)	6096678 (11%)	866284 (1%)
	Desocupado	11955368 (14%)	63431967 (74%)	9977663 (11%)
	Ocupado	629344 (12%)	1803225 (37%)	2415886 (49%)

Fonte: Produção do próprio autor.

Tabela 5 – Matriz de confusão para o banco de dados acumulado no *log* Volta da UFES de 2016.

		Inferência da rede		
		Desconhecido	Desocupado	Ocupado
Ground Truth	Desconhecido	53332217 (89%)	5416211 (9%)	834688 (1%)
	Desocupado	11385539 (12%)	70819386 (76%)	10656103 (11%)
	Ocupado	565147 (10%)	1484927 (28%)	3185782 (60%)

Fonte: Produção do próprio autor.

Tabela 6 – Matriz de confusão para o banco de dados não acumulado no *log* CENPES.

		Inferência da rede		
		Desconhecido	Desocupado	Ocupado
Ground Truth	Desconhecido	15205100 (70%)	6169729 (28%)	304219 (1%)
	Desocupado	1071517 (4%)	21142231 (86%)	2107809 (8%)
	Ocupado	68895 (5%)	572081 (49%)	518419 (44%)

Fonte: Produção do próprio autor.

As matrizes de confusão para os bancos de dados acumulado e não acumulado, nos *logs* da volta da UFES de 2016 e do CENPES estão representados nas Tabelas 4, 5, 6 e 7. Como se pode observar pelas tabelas, a maior parte dos acertos da rede estão concentrados nas classificações de desconhecido e desocupado, enquanto a maior parte dos erros está nas classificações de ocupado, nas quais boa parte das previsões incorretas vão para a classe desocupado. Vale ressaltar que apesar dos testes mostrarem uma baixa taxa de acerto na classe ocupado, uma análise qualitativa das predições mostra que a rede é mais precisa nas previsões de ocupado dentro das ruas, e que a maior parte das suas previsões ruins está nas regiões mais periféricas, onde não há muitas informações do laser. Esse teste mostra ainda que o acúmulo de nuvens apresentou resultados ligeiramente superiores para a previsão de célula ocupada.

Outro teste realizado foi a comparação da acurácia das predições da rede com a acurácia do *snapshot map* gerado para esses bancos de dados de teste. O *snapshot map* é o mapa de ocupação gerado pelo OGM da IARA sem acúmulo de mapas, ou seja, é o mapa instantâneo do OGM feito com a nuvem do LIDAR mais atual. A Figura 27 mostra um exemplo de *snapshot map* no *log* CENPES.

Tabela 7 – Matriz de confusão para o banco de dados acumulado no *log* CENPES.

		Inferência da rede		
		Desconhecido	Desocupado	Ocupado
Ground Truth	Desconhecido	15393665 (68%)	6860788 (30%)	252612 (1%)
	Desocupado	1297961 (5%)	21561271 (85%)	2386896 (9%)
	Ocupado	58061 (4%)	546676 (45%)	602070 (49%)

Fonte: Produção do próprio autor.

A partir desses testes as taxas de acerto do modelo neural não acumulado, acumulado e modelo probabilístico puderam ser comparadas. A Tabela 8 mostra as diferentes acurácias alcançadas nos bancos de dados de validação e teste. Em uma primeira análise se conclui que apesar na diminuição da acurácia entre os testes com banco de dados de validação e teste, a rede manteve relativo sucesso em generalizar seus resultados, já que a acurácia se manteve alta no set de teste. Em segundo lugar, pode-se confirmar que a rede com acúmulo de nuvens do LIDAR não apresentou resultados que comprovam de maneira explícita sua melhoria em relação à rede sem acúmulo de nuvens.

Por fim, destaca-se também que os modelos neurais conseguem uma melhor predição da totalidade do mapa em relação ao *snapshot map* probabilístico no teste. A acurácia do *snapshot map* em relação ao *ground truth* é esperada pela natureza do método. A comparação apenas tem objetivo de ilustrar a quantidade de células preditas pela rede neural em relação ao que é predito em apenas uma nuvem do LIDAR. Não necessariamente essa maior acurácia representa uma maior qualidade do mapa gerado.

Tabela 8 – Acurácia média dos modelos para os bancos de dados de validação e teste.

	neural map. não acum.	**neural map. acum.**	***snapshot map***
Set de validação	73%	73%	-
Volta da UFES de 2016	78%	80%	56%
Volta do CENPES	78%	76%	67%

Fonte: Produção do próprio autor.

4.3 Discussão

Como ficou comprovado pelos testes, a rede neural completamente convolucional LoDNN proposta por Caltagirone, Scheidegger e Svensson (2017) alcançou uma taxa de acertos boa se comparada ao *snapshot map* gerado pelo OGM (média de 78% do mapa neural contra média de 61% do *snapshot map*). Ainda, uma análise qualitativa dos resultados aponta para outro lado positivo desse modelo neural, que é a capacidade de realizar previsões em locais onde não tiveram muitos pontos do laser. A Figura 26, por exemplo, mostra que

para um dado instante do log CENPES tanto a rede com nuvens acumuladas quanto a rede sem nuvens acumuladas conseguiram fazer previsões pertinentes nos extremos das imagens, onde a representação dos dados de entrada possuem pouca ou nenhuma informação.

Por outro lado, o teste com matrizes de confusão mostrou que a rede apresenta uma baixa taxa de acertos de células ocupadas, ainda que o acúmulo de nuvens tenha melhorado ligeiramente essa situação. Em adição, o tempo de execução do *feedforward* da rede neural foi de em média 1,5 ms em um computador com 8 Gb de memória RAM e com placa de vídeo NViDIA GTi1060, sendo assim satisfatório para execução em tempo real desse modelo. De fato, se comparado com o tempo de execução excelente alcançado por Caltagirone, Scheidegger e Svensson (2017) de 18 ms, percebe-se que esse modelo neural tem um tempo de execução parecido. Portanto, com esse baixo tempo de execução tanto o mapeamento *offline* quanto o mapeamento *online* podem ser beneficiados pelo método neural apresentado.

Ressalta-se ainda que é necessário testes para comprovar a viabilidade do modelo neural para a navegação da IARA, visto que esse modelo não foi testado em um caso real de utilização, em conjunto com os outros módulos do carro.

5 CONCLUSÕES

Ao final deste trabalho, concluímos que o projeto de mapeamento em grade de ocupação utilizando redes neurais profundas atingiu o seu propósito de comprovar a viabilidade do modelo neural de inferências. Destaca-se que o mesmo atingiu uma maior precisão que o método probabilístico atual da IARA a partir das mesmas informações de entrada.

Dessa forma, os objetivos gerais foram alcançados, já que as etapas previstas nos objetivos específicos foram concluídas. Além disso, esse projeto abre novas possibilidades de estudos no tema, e se mostrou promissor para aplicações de mapeamento utilizadas pela IARA. Foi possível perceber que uma calibração mais refinada dos inúmeros parâmetros da rede convolucional utilizada poderia gerar resultados ainda melhores. Os estudos realizados mostraram que a implementação dessa rede na IARA é possível desde já para o mapeamento *offline* e *online*.

Por fim, propõe-se como projetos futuros primeiramente modificações e testes com de redes de segmentação diferentes para melhorar resultados e possivelmente reduzir o tempo de inferências. Outro tema interessante de continuação deste projeto é o estudo de melhoras possíveis para o banco de dados para treinamento, visando alcançar resultados mais generalizados e em ambientes mais dinâmicos. Ressalta-se principalmente o projeto de integração dessa rede ao controlador do carro, de forma a se confirmar a validade da mesma como gerador de inferências de ocupação para a navegação do veículo.

REFERÊNCIAS BIBLIOGRÁFICAS

ACADEMY, Data Science. **Deep Learning Book**. 2017. Disponível em: <http://deeplearningbook.com.br/>. Acesso em: 14 nov. 2018.

CALTAGIRONE, Luca et al. Fast LIDAR-based road detection using fully convolutional neural networks. **arXiv preprint arXiv:1703.03613**, 2017.

CLEVERT, Djork-Arné; UNTERTHINER, Thomas; HOCHREITER, Sepp. Fast and accurate deep network learning by exponential linear units (elus). **arXiv preprint arXiv:1511.07289**, 2015.

DARPA. **The Grand Challenge for Autonomous Vehicles**. 2005. Disponível em: <https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles>. Acesso em: 26 ago. 2017.

GEIGER, Andreas et al. Vision meets Robotics: The KITTI Dataset. **International Journal of Robotics Research (IJRR)**, 2013.

HAYKIN, Simon; MARTINS, Paulo. **Redes Neurais: Princípios e Prática**. 2. ed. [S.l.]: Artmed, 2007. ISBN 9788461504411.

JAIN, Anil K; MAO, Jianchang; MOHIUDDIN, K Moidin. Artificial neural networks: A tutorial. **Computer**, IEEE, v. 29, n. 3, p. 31–44, 1996.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2012. p. 1097–1105.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

MUTZ, Filipe et al. Large-scale mapping in complex field scenarios using an autonomous car. **Expert Systems with Applications**, Elsevier, v. 46, p. 439–462, 2016.

PASZKE, Adam et al. Automatic differentiation in PyTorch, 2017.

PEREZ, Luis; WANG, Jason. The effectiveness of data augmentation in image classification using deep learning. **arXiv preprint arXiv:1712.04621**, 2017.

REN, Shaoqing et al. Faster r-cnn: Towards real-time object detection with region proposal networks. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2015. p. 91–99.

SCHERER, Dominik; MÜLLER, Andreas; BEHNKE, Sven. Evaluation of pooling operations in convolutional architectures for object recognition. In: ARTIFICIAL Neural Networks–ICANN 2010. [S.l.]: Springer, 2010. p. 92–101.

SCHWARZ, Brent. LIDAR: Mapping the world in 3D. **Nature Photonics**, Nature Publishing Group, v. 4, n. 7, p. 429, 2010.

SIMARD, Patrice Y; STEINKRAUS, Dave; PLATT, John C. Best practices for convolutional neural networks applied to visual document analysis. In: IEEE. NULL. [S.l.: s.n.], 2003. p. 958.

THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. **Probabilistic Robotics**. 10. ed. [S.l.]: MIT press, 2006. ISBN 9788461504411.

TORCH CONTRIBUTORS. **Source code for torch.nn**. 2017. Disponível em: <`https://pytorch.org/docs/stable/_modules/torch/nn/modules/pooling.html`>. Acesso em: 2 dez. 2017.

WU, Haibing; GU, Xiaodong. Max-pooling dropout for regularization of convolutional neural networks. In: SPRINGER. INTERNATIONAL Conference on Neural Information Processing. [S.l.: s.n.], 2015. p. 46–54.

YU, Fisher; KOLTUN, Vladlen. Multi-scale context aggregation by dilated convolutions. **arXiv preprint arXiv:1511.07122**, 2015.